IFICIALLY INTELLIGENT AIR COMBAT SIMULATION AGE

THESIS

Daniel E. Gisselquist
Second Lieutenant, United States Air Force

AFIT/GCE/ENG/94D-04

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/94D-04

ARTIFICIALLY INTELLIGENT AIR COMBAT SIMULATION AGENTS

THESIS

Daniel E. Gisselquist
Second Lieutenant, United States Air Force

AFIT/GCE/ENG/94D-04

Approved for public release; distribution unlimited

AFIT/GCE/ENG/94D-04

ARTIFICIALLY INTELLIGENT AIR COMBAT SIMULATION AGENTS

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

Daniel E. Gisselquist, B.S.C.S. and B.S.M.S.

Second Lieutenant, United States Air Force

December 1994

Table of Contents

	Page
List of Figures	v
Abstract	vii
I. Introduction	1-1
1.1 Background	1-1
1.2 Previous AFIT Work	1-2
1.3 Research Goals	1-3
1.4 Assumptions	1-4
1.5 Scope	1-5
1.6 Summary	1-6
II. Literature Review	2-1
2.1 Distributed Interactive Simulation	2-1
2.2 Aerodynamic Aircraft Models	2-1
2.3 Relevant AI Methods	2-2
2.3.1 Reaction	2-2
2.3.2 Deliberation	2-3
2.4 Similar Autonomous Agent Research	2-4
2.4.1 MAXIM/CAP	2-4
2.4.2 PDPC	2-6
2.4.3 TacAir Soar	2-7
2.5 A Project History	2-8
2.6 Summary	2-9

Page

III. Methodology 3-1

3.1 Environment Generation 3-1

3.1.1 DIS Network Processing 3-2

3.1.2 Local Models 3-3

3.2 Agent Reasoning 3-5

3.2.1 Reactive Reasoning 3-6

3.2.2 Deliberative Reasoning 3-9

3.3 Experiment Design 3-11

3.4 Summary 3-11

IV. Implementation 4-1

4.1 Environment Handling 4-1

4.1.1 Network Reading 4-1

4.1.2 Model Manager 4-2

4.2 Phase processing 4-3

4.2.1 Phase Descriptions 4-3

4.2.2 Event Types 4-4

4.2.3 Event Buffers 4-5

4.2.4 Event Control 4-6

4.3 Reaction 4-6

4.3.1 Threat Monitoring 4-6

4.3.2 Maneuver Selection 4-6

4.3.3 Weapon Utilization 4-7

4.4 Agent Planning 4-8

4.5 Flight Control 4-10

4.5.1 Maneuvers 4-10

4.5.2 Building Blocks 4-13

4.5.3 Throttle and Stick 4-15

4.6 Summary 4-16

Page

V. Analysis 5-1

5.1 Effectiveness 5-1

5.1.1 Air-to-Air Combat 5-1

5.2 Threat Avoidance 5-5

5.3 Solution Timing 5-7

5.4 Summary 5-8

VI. Conclusions 6-1

Appendix A. Planning Partial Solutions A-1

Vita VITA-1

Bibliography BIB-1

List of Figures

Figure		Page
1.1.	Intelligent Agent Components	1-2
1.2.	Problem areas addressed	1-3
2.1.	The Work of MAXIM/CAP	2-5
2.2.	PDPC Contribution	2-7
2.3.	TacAir Soar Focus	2-8
3.1.	Network Reading	3-3
3.2.	Local Models	3-4
3.3.	Reactor Decomposition	3-7
4.1.	Six State Description Variables	4-6
4.2.	State Description Evaluators	4-7
4.3.	Possible Paths from a Cell	4-8
4.4.	Maneuver Design	4-13
5.1.	The Non-Jinking Bogey Scenario	5-2
5.2.	Overshoot against the Jinking Bogey	5-3
5.3.	Agent vs Agent Stalemate	5-5
A.1.	Agent vs AAAs, T=0 min	A-2
A.2.	Agent vs AAAs, T=2 min	A-2
A.3.	Agent vs AAAs, T=4 min	A-2
A.4.	Agent vs AAAs, T=6 min	A-3
A.5.	Agent vs AAAs and Drones, T=0 min	A-3
A.6.	Agent vs AAAs and Drones, T=1 min	A-3
A.7.	Agent vs AAAs and Drones, T=2 min	A-4
A.8.	Agent vs AAAs and Drones, T=4:30 min	A-4

Figure Page

A.9. Agent vs AAAs and Drones, T=5:40 min A-4

AFIT/GCE/ENG/94D-04

Abstract

This work is concerned with developing techniques that may be used in the creation of artificially intelligent air combat simulation agents. It demonstrates, by example, that it is possible to integrate both reactive and deliberative behaviors within a highly dynamic real-time combat environment. The culmination is an intelligent agent which exhibits many of the behaviors necessary in such a combat situation.

ARTIFICIALLY INTELLIGENT AIR COMBAT SIMULATION AGENTS

I. Introduction

1.1 Background

The Advanced Research Projects Agency (ARPA), in an effort to improve military simulations, has proposed a standard for Distributed Interactive Simulation (DIS). This network protocol will allow dissimilar applications to communicate simulation information. In the field of air combat, this will allow pilots in remote simulators to fly in a common simulation.

The success of this standard is dependent on applications being able to work within that standard, as well as a large variety of Computer Generated Forces (CGFs) to complement interactive forces. CGFs are needed to help simulate the large combat scenarios that can only occur in wartime. By augmenting human forces with large numbers of computer forces, these large combat scenarios can be created and tested without involving a proportionately large number of people.

This work explores the design requirements of artificially intelligent air combat simulation agents. These agents will be used to provide air-to-air forces. Since pilots will train against them, and build the reflexes needed to conquer them, these air CGFs need to be as realistic and capable as possible. Currently, such agents are easy to recognize. Once recognized, they are easily destroyed. (9) The goal of this effort is to develop techniques that can be used to make artificially intelligent air-to-air combat simulation agents more effective, more deadly, and harder to spot.

In order to understand an effort in this domain, it is helpful to break it down into three basic parts: the intelligence portion, the simulation models portion, and the network interface portion. In the real world, this would correspond to the pilot, the aircraft, and the rest of the world. Figure 1.1 shows the various components of each of these three parts. The intelligent agent is concerned with controlling its aircraft (the flight model)

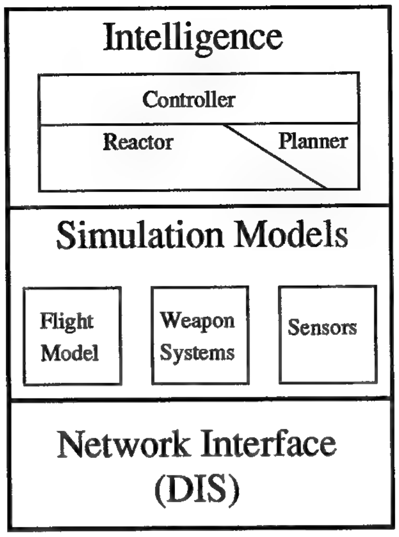

Figure 1.1 Intelligent Agent Components

and the weapon systems onboard that aircraft. It receives information through its sensors. The sensors and models then coordinate world state information through the network interface. Forms of this figure will be used throughout this document to highlight research and contributions into different subsections of such CGFs.

This effort will focus primarily on the requirements of systems necessary for 1v1 visual guns-only combat. It will also take a look at the possible methods for integrating reaction and planning to navigate a complex battlefield. It is hoped that these techniques will prove valuable for future real-time artificial intelligence applications.

1.2 Previous AFIT Work

This current project is a third generation project at AFIT. It is preceded by the MAXIM/CAP, and PDPC projects. Each of these projects created artificially intelligent air combat simulation agents which could fly against each other.

MAXIM/CAP was a 1992 class project at AFIT (5). It demonstrated two separate approaches to developing combat simulation agents using the proposed DIS standard. One technique involved embedding the intelligence within a LISP procedure-based representation scheme, while the other technique involved using the C Language Integrated

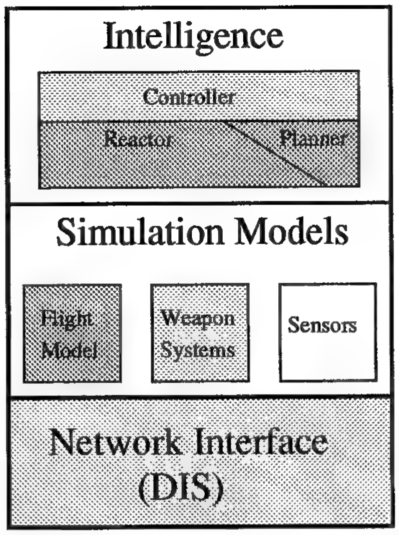

Figure 1.2 Problem areas addressed

Production System (CLIPS). These systems demonstrated the feasibility of creating air combat simulation agents as well as some of the problems.

The PDPC project was initially intended to be an extension to the MAXIM/CAP. This work, produced by Capt Dean Hipwell and CPT George Hluck, abandoned the DIS capability and attempted to focus on a proper form of knowledge representation for the air combat knowledge that it contained (6) (7). This research contributed a knowledge representation structure of phases and maneuvers upon which information could be added.

1.3 Research Goals

The goal of this effort is to develop techniques that can be used to make artificially intelligent air combat agents more effective. To do this, I intend to show that it is possible to integrate reactive and deliberative reasoning within a highly dynamic, real-time environment. Previous work has not accomplished this goal within the domain of real-time air combat. Figure 1.2 highlights those topics of intelligent agents which this research addresses. While the controller, the weapons systems, and the network interface subsystems are addressed, particular attention is paid to the aircraft control, air combat reaction, and threat avoidance planning subsystems.

Previous air combat research at AFIT has not made an effort to capture the realism involved in flight. Aircraft have been allowed to slow down without stalling, and to turn without losing energy. Specifically, this interplay between speed, altitude, and turn radius inherent in flight has not been modeled. By using an aerodynamic flight model, I hope to discover and explore new parts to the air combat problem. A credible flight model will precede credible actions. In order to keep the model realistic, the granularity of the simulation time will need to fall within the model's limits. As a result, actions will need to take place within that limited amount of time. Failure to respond in a timely manner to outside events will result in the termination of the agent. An example of such a situation is a missile warning.

In order to respond to events fast enough, the agent will have a reactive ability. This work will demonstrate how such reaction can use aircraft control techniques to make the model respond appropriately in all situations.

Finally, the limitations of reaction necessitate some form of planning capability in an intelligent agent. This lack is one of the reasons why current artificial simulation agents are so easily spotted. This is where I would like to add to the current capabilities. By programming the agent to plan and re-plan in flight, the agent will be able to dynamically determine a best course of actions to follow. I will address the idea of a path planner to supplement the agents reactive capabilities.

1.4 Assumptions

In order to reduce the domain of air combat to something more solvable, two assumptions will be made. The first is that combat will be limited to one-on-one scenarios and that agents need not concern themselves with the possibilities of fratricide. This eliminates the need to check all active munitions for those which might hit the agent. This also eliminates the need to check for possible friendly entities in the line of fire from the agent to its target. In small simulations, the probability of an entity getting caught in the cross-fire between two others is small. However, as larger simulations are built these checks must be added for realism. The second assumption is that the best knowledge representation and structure for a human is not necessarily the best representation for the computer. This work is not intended to recreate the errors as well as the proficiencies of human pilots, but rather to create the best simulated air combat agent possible. This assumption is contrary to the fundamental assumptions of Soar (11). In order to be indistinguishable from human pilots, such errors will need to be modelled in future work.

Finally, no level of modelling will ever be enough to accurately model what happens in a combat situation. The true number of things that a combat agent must interpret and reason about may never be known. Examples of these things include visual processing, speech processing, signal processing, and a host of other modelling problems. These will be treated as separate problems, of which the solutions will be assumed known. Future research in this domain will need to include the limitations associated with these solution processes.

1.5 Scope

The scope of this research effort is limited to a very narrow portion of air combat. As a result, the fighter combat scenarios will be limited to one-on-one combat with guns only. One-on-one combat will demonstrate the reflexes required while eliminating the need for more complex priority decision making. To demonstrate that a planner may be integrated, anti-aircraft artillery and drones will be used to give the agent something to avoid. Anti-aircraft artillery will demonstrate the agent's capability to plan around fixed objects, while the drones will exercise the agent's capability to plan around moving objects. Within this limited scope, however, I will attempt to create an agent that can successfully fly a simulated aircraft through this combat environment.

While developing planning and reaction integration techniques, it is not my intention to create the perfect combat reactor, nor is it my intention to create the perfect planner. I do intend to demonstrate how a planner and a reactor can be combined to function within an air combat environment. I also intend to demonstrate that this integration will make agents more capable.

1.6 Summary

ARPA has requested research in the development of artificially intelligent combat simulation agents to aid in the creation of large scale combat scenarios. This research effort has the goal of developing techniques that can be used to make simulated air combat agents more effective.

II. Literature Review

CGFs are needed within the DIS environment. This work will attempt to create an agent within that environment that is capable of both reaction and planning, and to explore the difficulties and consequences of such an integration. Therefore, before looking at previous air combat agent research, a review of the simulation principles and relevant artificial intelligence methods is appropriate.

2.1 Distributed Interactive Simulation

Although DIS is a proposed simulation standard, applications are already being built to its specifications. The draft spells out an architecture for future computer combat simulations:

> DIS is a time and space coherent synthetic representation of world environments designed for linking the interactive, free play activities of people in operational exercises. The synthetic environment is created through real-time exchange of data units between distributed, computationally autonomous simulation applications in the form of simulations, simulators, and instrumented equipment interconnected through standard computer communicative services. The computational simulation entities may be present in one location or may be distributed geographically. (3)

To be compatible with the DIS environment, it is necessary to create a computationally autonomous simulation application. As long as the application follows the communications protocols spelled out in the standard, it should be able to communicate with all other DIS applications. The end result can be a rich and diverse combat environment involving large numbers of participants.

2.2 Aerodynamic Aircraft Models

Most of the artificially intelligent work dealing with aircraft works with fairly simple aerodynamic models. Here at AFIT, our models have centered around point-mass systems given allowable turn rates and longitudinal accelerations. Flight has even been simulated in spaces without gravity or well-defined units (6). This is not realistic in a flight situation. Pilots must routinely juggle interacting factors such as altitude, turn-rate, and speed.

Stevens and Lewis have published an F-16 model created from wind-tunnel data on a scale model (15). This model does not completely describe all of the characteristics of a standard F-16 in flight: it is not accurate past 0.6 Mach, the leading edge and trailing edge flaps have been removed, stall is not modelled well, and the aircraft does not have enough pitching moment for angles of attack past 25 degrees. Despite these shortfalls, it is a model that is more realistic.

This model also imposes requirements to the reasoning agent. The model accepts as inputs throttle and stick commands. Instantaneous thrust in any direction is not possible; pulling back on the stick causes drag to increase. The simulation application must also integrate the model at time steps less then a tenth of a second for accuracy. As long as the simulation can compensate for these new problems the model should be close enough to the real thing to fly against and be evaluated by pilots in other simulators.

This model will be used to model aircraft capabilities in this research.

2.3 Relevant AI Methods

The artificial intelligence required of an independent agent has taken on two rough forms: reaction and deliberation. The distinction between these two methods can easily become blurred. They both attempt to solve what I will term the *intelligence problem.* That is, given the condition of the world and a set of actions determine what action the agent should perform. Two approaches, reaction and deliberation, to solving the intelligence problem are described below.

2.3.1 Reaction. Reaction functions can described as a set of state-action pairs called a universal plan. This universal plan, which ideally consists of the best actions for any probable state, is pre-computed. A pattern-matching engine then matches the current state with a state in the universal plan, and executes the associated action. Agre and Chapman have demonstrated that simple functions of this type can yield complex behaviors that look as though they are well planned and thought out (1).

Schoppers defines a universal plan as a set of state-goal-action combinations (12). Thus when the intelligent agent matches a world state with a stored state, it performs the associated action. By adding the goal to the state-action combinations, Schoppers has deftly allowed the universal plan to solve more than one type of problem by simply changing the goals.

Since complete enumeration of all possibilities can quickly overwhelm a computer, shortcuts are made. Possibilities include state descriptions that match large sets of world states, and partial reaction functions that do not know how to respond to every situation. Mitchell describes such a partial system (10). By planning state-action pairs when necessary, and using them to modify the set, a robot can become increasingly reactive and adaptive.

2.3.2 Deliberation. In combat, constant reaction is not enough. If such a constant universal plan were known, the agent's opponent only needs to find one case where the action in the plan is inappropriate for the given state (2). The success of a universal plan depends on the adversary not knowing it. Therefore, some mechanism must keep the universal plan unpredictable and at the same time correct under new situations.

Specifically, deliberation is a matter of attempting to solve a problem. Inside a universal plan, expected problems are pre-solved and the associated solutions are stored with them. Deliberation attempts to solve those problems that have not been solved previously. Although deliberation can take on several forms, for the purposes of this research, planning will be considered.

"The planning problem may be characterized as the problem of determining an ordered sequence of actions which when executed from a given initial state will achieve a given goal."(4) Classical planning traditionally attempts to string operators together from some initial state to achieve a desired goal state. The decision of which operator to apply when is made through a matter of searching through all possibilities.

The key difference between following a plan and reacting to the world is whether or not the agent makes its decisions based on previously generated knowledge (11). An agent may look at all of the possible outcomes to a given action and decide on appropriate responses to each ahead of time. As such, the agent has created a portion of a universal plan. This universal plan may include contingency plans as well by anticipating possible failure states and calculating appropriate responses. In a combat domain, the world is very unpredictable. Therefore, "the planner must anticipate possible situations and predetermine its reactions to those situations."(12)

This unpredictability yields another problem as well: time. Problems may change during attempts to solve them. Previously computed solutions may become outdated and inappropriate. Therefore, both the planning process and the database of plans that it produces are inherently time-dependent in this domain.

Many attempts at time-dependent planning, while significant, have not dealt with the problems uniquely associated to the air combat domain. Although finding the fastest path from point A to point B may easily involve stopping to think (14), aircraft in general cannot do this. This leads to problems with classical methods which begin with characterizing the problem and then attempting to solve it. The flight problem does not stay characterized for very long. Therefore, the initial state of the plan cannot be known ahead of time.

In general, the difference between reaction and deliberation tends to be how the intelligence problem is solved. If it is solved by anticipating problems and calculating their solutions, it is termed deliberation. If it is solved by storing the solved problems, and then matching the current problem to the stored problem to determine an appropriate action it is termed reaction.

2.4 Similar Autonomous Agent Research

This thesis effort is preceded by several other similar efforts, both at AFIT and elsewhere. Significant among these are the MAXIM/CAP project, PDPC, and TacAir Soar.

2.4.1 MAXIM/CAP. MAXIM/CAP was a class project at AFIT (5). It was designed "to develop autonomous agents which exhibit appropriate behaviors for simulated air combat." After it began, the project split into two separate approaches: MAXIM and CAP.

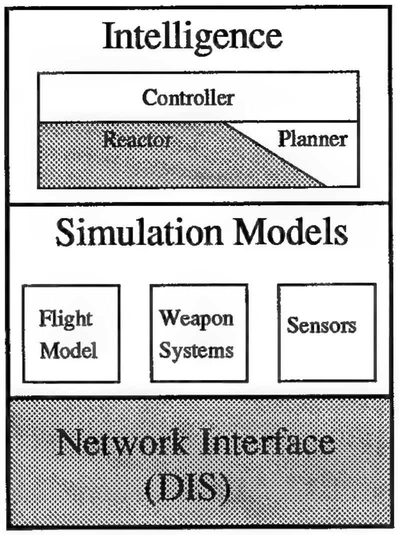

Figure 2.1 The Work of MAXIM/CAP

MAXIM was a procedurally based simulation built in Lisp. It had a standard simulation architecture consisting of an inner loop which repeatedly read the network to update external entity representations, updated the simulation manager, and then updated all of the local entities. This process incurred a slight lag which was not quantified in the report.

CAP was very similar to MAXIM in many respects, with the exception that it was built within the rule-base paradigm. The agent flew by determining what state matched the current one and accomplishing the actions associated with that state. The aircraft was controlled by calculating points to fly to which the model then attempted to accomplish. This approach had some overhead problems as well: "CLIPS is an interpreted environment and as such, a significant amount of overhead is incurred while executing CLIPS coded functions. As the aero-model we have developed becomes stable, translating it into C and integrating it into the CLIPS environment should provide a significant increase in the update cycle." (5) This translation was never accomplished.

Figure 2.1 highlights the contributions of MAXIM/CAP. These projects demonstrated the feasibility of using a rule-based paradigm and a procedure-based paradigm to develop air combat simulations. The resulting agents, however, are incapable of higher-level reasoning and, as such, they tend to be highly simplistic. The need for a deliberative intelligence becomes more pronounced the more such agents are used.

2.4.2 PDPC. Pilot Decision Phases in CLIPS, (PDPC), "... is a rule-based reactive system where objects act as autonomous agents who can maneuver in an air combat situation." (7) Like CAP, PDPC was written in CLIPS. Its agent architecture consisted roughly of two parts: phase and maneuver. The phase was the agent's current high-level state, while the maneuver was its current flying goals. First the agent would look for the conditions necessary for changing phases. For example, when the aircraft identified a target while it was cruising, it would transition from *cruise* phase to *attack* phase. Once it had determined the appropriate phase, the agent would determine a maneuver and execute it.

By breaking the agent's decisions into distinct phases, it became possible to separate these phases into separate modules. Each module had its own rules, facts, and agenda. Modules could share rules or facts with other modules, but this was not necessary. If a module ever ran out of rules to fire, it automatically shifted back to the main module. During the course of an update, CLIPS would enter the active modules for each agent and reason. Once all agents had reasoned, control returned to the main module and updated the aircraft. Unlike CAP, PDPC was written entirely in CLIPS.

PDPC also included the development of several Titan modules, named after the Titan report which influenced their development (8). These modules were written to accomplish the higher-level decisions necessary during flight combat. Examples of these decisions include deciding whom to attack, deciding how to set the radar, and deciding what formation and attack strategies to use. This emphasis on weapon system support decisions can be seen in Figure 2.2.

Despite the initial goals of PDPC, it was not compatible with the requirements of the DIS environment. The most difficult part of this problem was that, although it was designed with timing considerations in mind, it did not attempt to operate within a real-time environment. The basic time unit in PDPC was an iteration which had no connection to the passage of time in the rest of the world. PDPC also suffered from other problems since it was written in a unit-less environment. Even so, PDPC took the initial results of MAXIM and CAP and continued the research into the development of reactive air combat agents by quantifying weapon systems decisions within a reactive model.

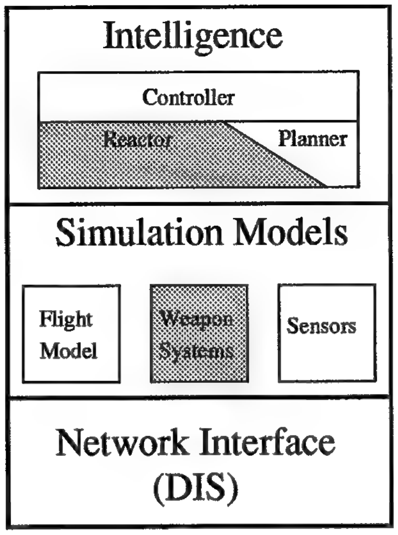

Figure 2.2 PDPC Contribution

2.4.3 TacAir Soar. Understanding the basics of Soar helps to explain the approach taken by TacAir Soar. The architecture of Soar has been designed around some basic assumptions. Soar has assumed that general intelligence can be studied in humans and computers in the same way. As such, Soar is a model of human cognition first, and a means of teaching computers to accomplish tasks second. Soar projects are also designed to push the Soar architecture to extreme limits to evaluate how much it can do (11).

The focus of TacAir Soar is to create pilot agents that fly simulated beyond visual range combat (BVR). Recognizing the need for an ever-expanding universal plan, a necessity in combat (2), TacAir Soar attempts to expand the universal plan in flight. It is designed to be able to adapt to new and unexpected situations by learning from experience. By using an architecture that can accomplish all forms of planning as well as reaction, the authors hope to be able to thoroughly integrate planning and reaction into the real-time problem solving. TacAir Soar agents must deal with limited information, reason about that information, and attempt to make appropriate deductions about the world. (9)

To date, Soar agents have been taught how to fly several missions on several different aircraft. They are capable of engaging in 1v1, and 1v2 combat. Current research is looking into natural language interfaces, planning, and agent modelling as well as the issue of multiple interacting goals (16). This focus can be seen in Figure 2.3. Of some note is the

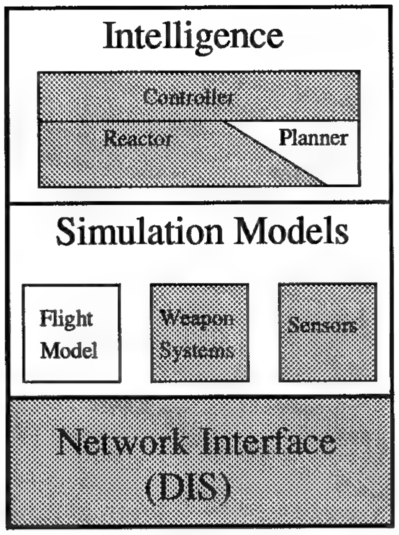

Figure 2.3 TacAir Soar Focus

fact that these Soar agents have yet to be able learn from their combat experience. To date, they have not been able to solve the problems involved with learning in this domain, nor have they integrated a planning capability into their model. Their universal plan is, therefore, static.

The focus of TacAir Soar is slightly different from that of this research. First, in the domain of air-to-air combat the agents in TacAir fight beyond visual range. This research looks at close range combat. Second, TacAir Soar has been unable to expand their Universal plan, despite their attempts. Plan expansion is one of the goals of this research effort.

2.5 A Project History

This current research project has progressed through several revisions. While the incremental approach was applied to many different problems, there were three basic designs attempted.

The first program demonstrated that an aircraft model that included the aerodynamics of an aircraft could be controlled. It was written in Ada with the understanding that a task structure would be most appropriate for describing what the aircraft could do.

This effort had to be abandoned when the Ada run-time environment available at AFIT at the time was unable to support the overhead of such a venture. This initial program was not built to be intelligent. As such, it could only follow pre-programmed scripts. It proved the feasibility of the control design which will be discussed in section 3.2.1.5.

The second program was written in C to use the same aircraft model. PDPC was used to control the aircraft. Given the nature of an aircraft model which needed constant control, it no longer made sense to control the aircraft through rules in the same manner. CLIPS maintains only one context and such constant rule firing would prohibit any other rules from firing. Therefore, the bottom-half, and most complex, portion of the PDPC code was rewritten in C, as suggested in MAXIM (5).

This design, which included the introduction of CLIPS to a changing environment, had to be abandoned for several reasons. The knowledge control reasoning built into PDPC was ineffective in an environment that changed while it was attempting to chain rules. As a result, when rules started to chain, the changing situation never let the chain increase longer than a couple of links before starting over. Finally, the CLIPS update rate was neither fast nor reliable. It could be as fast as four to five times a second, but it could slow down to a second and a half per update. Unable to resolve these complications, the original PDPC programs and code were abandoned or rewritten completely within procedural paradigm. The reactive decisions were re-implemented within the procedural paradigm.

2.6 Summary

Presented in this chapter are various artificial intelligence schemes which are useful for developing combat agents, as well as some previous work which used these schemes. The next chapter, Methodology, will describe the methods which I chose to employ for this endeavor.

III. Methodology

There have been many steps along the way to developing an intelligent agent. This chapter will present the major problems, as well as describe the solution technique implemented. For space and time considerations the complete design from top to bottom will not be presented. Instead, those portions of research most important to the design and implementation of artificially intelligent combat simulation agents will be shown. Therefore, starting with the overall methodology, this chapter details the major portions of the developed simulation and their design. It includes the basic agent design, as well as the design of its subcomponents. The effectiveness of this design will be demonstrated and critiqued in subsequent chapters.

While several design methodologies may have applied, one was used predominantly: incremental design with a bottom-up approach. By building reusable code at the bottom level, the upper level could be, and has been, redesigned and re-implemented as necessary with as little additional work as possible. The incremental approach has been used for the sole reason that I did not know ahead of time the problems which would be encountered in an effort of this type. Were this project to have been started from scratch with all of the knowledge learned thus far, a different design methodology may have been more appropriate.

3.1 Environment Generation

Common throughout all of the software designs has been a fairly consistent environment design. The environment is the connection the agent has to its world. It is also the very top-level of the simulation. It supports multiple communicating entities across a network of sites using the DIS communications protocol. Simulation generation can be effectively split into two subparts. The first involves keeping track of what is going on locally with a variety of simulation models. The second involves keeping track of what else is going on by processing the DIS network packets. By repeatedly processing a buffer of packets from the network, and then updating all the local models, a simulation environment may be maintained. The faster this is accomplished, the faster the agent will be able react to its environment and the greater the model fidelity will be.

3.1.1 DIS Network Processing. Although the DIS protocol contains a whole suite of packet types, and to be fully DIS compliant all types should probably be supported, implementation of only a few such packets yields enough information to accomplish a limited fighter combat scenario. The packet types that the simulation pays attention to are entity state, fire, and detonate protocol data units (PDUs).

The network connection itself attaches to a daemon which reads PDUs from the network and buffers them upon arrival. The simulation application periodically empties this buffer to determine the state of the external world. Packets created by an application using a different version of DIS or a different exercise ID are filtered out immediately. Following this initial filtering, the packet is filtered based upon its type and sent to an appropriate handling procedure based upon its type.

Entity State PDUs, or ESPs, are essential for knowing where other entities are. ESPs are broadcast for all entities at a minimum rate of one every five seconds. If an entity fails to meet its five second deadline, it is removed from the simulation. In the meantime, the entity's position is determined based upon several dead-reckoning parameters located in the ESP. The entity retransmits this packet once its location and orientation parameters no longer fit within an inertial model. In order to maintain an accurate world model a storage place must be kept for each entity's most recent ESP. This storage place is the world model shown in Figure 3.1. A received ESP is sent immediately to this storage area. The storage area can then be queried for information. Position, velocity, and orientation parameters are projected with a dead-reckoning model when read. Thus the agent can request, from the world model, any ESP information about any object in the external environment.

The other type of PDU processing handles fire and detonation PDUs. Upon receipt of a fire PDU, the target agent is informed. In an actual combat scenario agents would need to notice all munitions and then decide which were important. For the purposes of this work, it is assumed that this can be done. Since the agent does not process such information, it has no need of the information located in the detonation PDU. Detonation

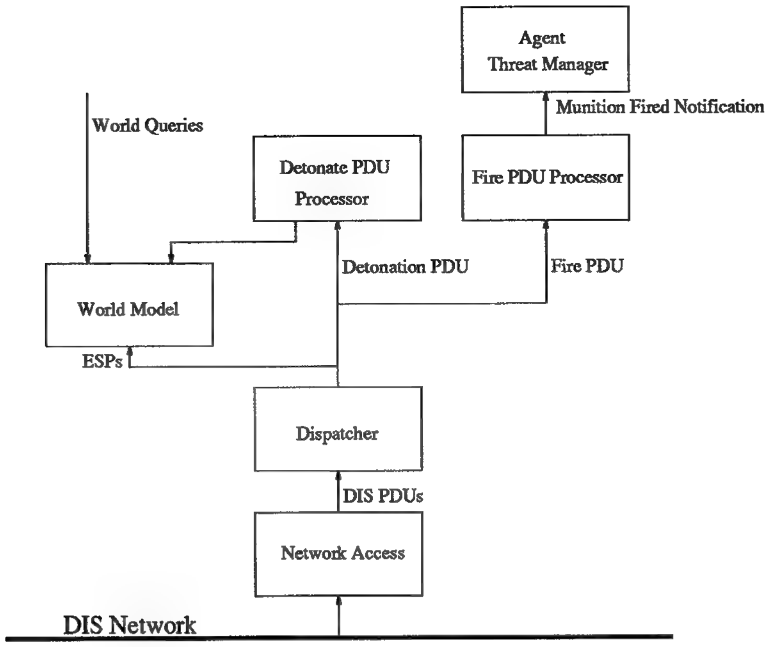

Figure 3.1 Network Reading

PDUs travel instead to the model object. This object determines an appropriate amount of damage and then adjusts its behavior based upon that damage. This adjustment in how the model behaves forces the agent to deal with a vehicle which does not respond the way it expects it to. Recognition of this problem has not been implemented. Since the agent does not perform failure detection and correction, this will hasten the agents destruction. Then, upon detonation, the munition entity ceases to exist in the world model.

By listening to the PDUs as they travel across the network, processing them, notifying the agent and allowing for queries, the simulation is semi-DIS capable. Further DIS capabilities such as re-supply, collisions, repairs, and data-queries are left for future efforts. The lack of transmit and receipt PDUs also keeps the agent from engaging in simulated electronic warfare–a very real part of modern battlefields.

3.1.2 Local Models. As stated earlier, the simulation maintains local models. Such models must exist for the aircraft as well as each of the munitions that it may use. Three types of models are currently implemented: agents (F-16s), AAA sites, and ballistic munitions (20mm cannon shells).

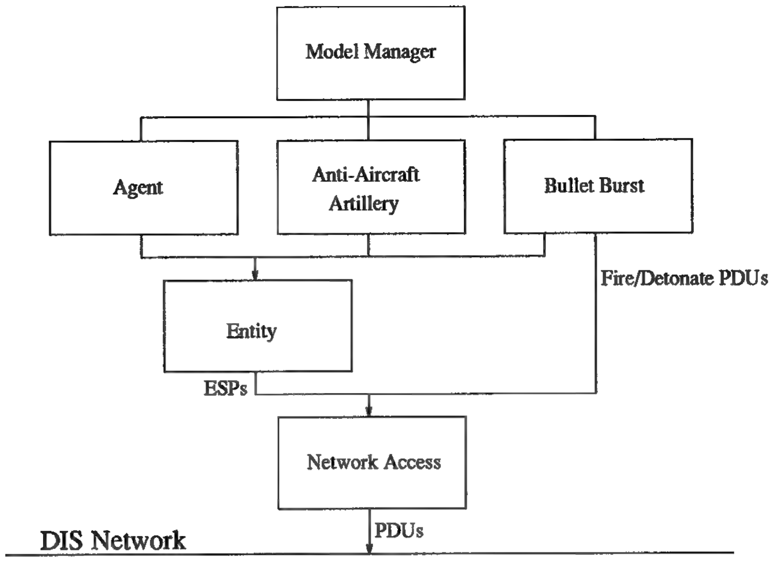

Figure 3.2 Local Models

The aircraft model used for the agent can be found in <u>Aircraft Control and Simulation</u> (15). As an aircraft model, it accepts inputs of throttle and stick and produces a state output including location, velocity, and orientation information. Since the model is non-linear, in order to maintain solution integrity, it must be updated a minimum of ten times a second. Since this is not always possible due to system load fluctuations, when the update time interval is greater than a tenth of a second, the state is integrated in intervals of a tenth of a second at a lower resolution until the updates have caught up. Once the update is finished, the simulation sends an ESP packet across the network for other simulation applications.

The ballistic munition model is perhaps the simplest model of the simulation. When the agent elects to fire its cannon, a burst is generated and a fire PDU is issued. Following this, ESPs are broadcast identifying the burst location. Bullets start at a firing velocity plus the aircraft's velocity and fall with gravity. When the burst comes within a short distance of its target, a detonation PDU is broadcast by the simulation. The location of the detonation is calculated as the point where the munition would pass closest to the target.

Damage is assessed by utilizing the distance of the weapon detonation to the model. Direct hits will destroy the aircraft, and the damage diminishes the farther out the impact occurs.

These models are sufficient for engaging in a DIS simulation of air combat. The next step is to place a reasoning agent within this environment. The agent will be evaluated on how well it acts and reacts to surroundings in this environment.

3.2 Agent Reasoning

Now that this environment has been defined and created, it is possible to place an agent within it. A look at the reasoning process used in PDPC will be instructive on how to accomplish this goal.

In PDPC, a chain of rules needed to fire for each agent, each iteration, resulting in the actual position update. PDPC contained several such chains, and the current situation determined which one fired. For example, there were chains within the Titan modules that determine who to attack, how to attack, and how to set and adjust various aircraft systems. Phase modules had several links in these chains as well. The aircraft first determined whether or not it should stay in the current phase or, if not, what phase it should enter into. Then it decided how to accomplish its task, followed by determining amounts of throttle and stick to use. These last two decisions were made by firing one or more rules, depending upon the decision complexity. If a new situation was detected, there were rules in the Titan decision files to analyze the new situation. Since PDPC was time independent, its agents always had time to reason through these chains.

In contrast, when dealing with a dynamic, real-time environment, the agent will not always have time to reason. Some reasoning processes need to be begun, killed, suspended, and/or resumed throughout the mission. Many of these processes must meet some response-time criterion. For example, a pilot cannot ignore his throttle and stick while flying, nor can he ignore the attacker that has gained a significant firing advantage on his tail. The instantaneous decisions will be accomplished with a reactive process, while the ones where the agent spends time reasoning will be accomplished with a deliberative process.

3.2.1 Reactive Reasoning. Most of the reactive reasoning used has a time requirement associated with it. Reasoning about a given event must be accomplished within the given time limit or risk catastrophic failure. Five types of reactions have been used for air combat agents in this research. These are phase, maneuver, weapon, threat and control reactions.

3.2.1.1 Phases. PDPC used the Titan research report (8) to define its phases. Phases are high-level states. For example, when the agent is in the *take-off* phase, the agent is concerned only with those problems that plague an aircraft during take-off.

Since there are often several stages to a given phase, each phase description also includes a sub-state within that phase. This is important when there are several steps to accomplish a given phase. For example, *landing* depends upon entering a pattern, approaching the runway, flaring and touching down. By allowing for sub-states, a highly procedural action can be encoded when necessary by explicitly describing state stages and their transitions within a given phase.

Phases changed in PDPC whenever certain preconditions were true. Upon recognition of a certain event the agent must respond to it in a state dependent manner. This PDPC structure has been replaced with an event-driven paradigm in this research. Since the design no longer rests within a rule-base paradigm that encodes events explicitly, events must be recognized and processed. The resulting design involves the phase module repeatedly requesting events, and reacting to them accordingly. Thus the agent, based upon sensed stimuli, will traverse its state space in a way appropriate to accomplish its mission.

3.2.1.2 Maneuver Selection. In a combat situation, the pilot has very little time to react to an opponent's moves and the opponent cannot be expected to leave mistakes uncorrected. Therefore, the agent should be able to change its behavior quickly based upon the opponent's move and the given scenario. A universal plan of maneuvers accomplishes this task. As described in section 2.3.1, there are two parts to any universal plan: the set of states and the set of associated actions (see Figure 3.3).

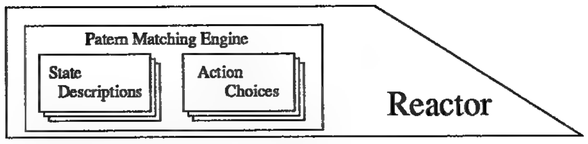

Figure 3.3   Reactor Decomposition

The set of states are used to describe the situations under which the associated actions are appropriate. Once the agent matches the current state with the state that fits it best from its set, then it knows how to act. If there are a large number of possible states, and a significantly large number of distinct reactions, then it makes sense to encode these states explicitly and do a straight match from state to operator if this is possible. On the other hand, if this set is sparse it makes more sense to calculate a distance function to determine which state is closest to the current one. Given the set of variables that make up a state, a description of that state will consist of several parts for each variable. For each variable which is important to the state description, some ideal comparison should be given followed by how important it is that that description is met exactly. Then an evaluator checks all of the state descriptions and picks the one which is the best match.

Associated with every best match is an action. While it is possible to select throttle and stick inputs at this level, doing so would require an enormous state table including every reasonable state the aircraft could be expected to get into. This additional description would include much more information about the situation than just information about the opponent and the combat situation relationships. The prohibitive size of such a state table limits the practicality of this approach. In order to keep the state table size to a minimum, the actions associated with a state are functions mapping the aircraft's state and the relevant state of the world to throttle and stick commands. These actions are memoryless, allowing an action to be selected and de-selected at any time. The actions are then used to achieve certain goals in the combat situation. These action functions shall be termed maneuvers.

While this routine will work in a combat scenario, and the recognition of a combat state can be accomplished very quickly, this efficiency is not necessary for normal flight. Therefore, this form of maneuver selection is something that can be turned on or off.

When it is not being used, maneuver selection can be accomplished by the phase and state transition routines. In order to do this, maneuvers must recognize their completion and signal this event to the phase structure so that it knows it can continue in a procedural model with whatever it was doing. While this does not address the question of how to handle the aircraft in an emergency, the perfect world of simulations and aircraft with no price tags does not need ideal emergency flight handling. If emergency handling were necessary, an approach similar to that used in combat maneuvering, based upon different state parameters, would be a good beginning.

3.2.1.3 Weapon Utilization. Given any target, there are two basic weapon choices to be made: what to fire, and when to fire. Weapon selection can be accomplished in the same manner as maneuver selection by describing the ideal firing solution for each weapon and then selecting the state that matches it the best. Knowing when to fire is dependent upon the current weapon, and whether or not the weapon is aimed.

The first part of the decision is simple: if the weapon cannot reasonably be expected to hit the opponent, do not fire. In the case of guns, the weapon can be reasonably expected to hit if bullets will pass within a small range of the other aircraft. Although there are situations in which pilots will fire just to scare an opponent, such behavior is not implemented. Thus the agent calculates how close its bullets would come to hitting its opponent. If they do not come within a given range, it will not fire.

The final piece of this decision is the question of when to quit firing. I chose to have the aircraft continue firing as long as the other aircraft could pose a threat. This answer works well in a combat environment with few opponents, but it may not work as well when the agent must conserve resources or rapidly switch between threatening opponents.

3.2.1.4 Threat Recognition. Threat recognition can be accomplished with the same basic method as weapon selection. Determining how close the aircraft is to the firing envelopes of all the weapons of a particular enemy will inform the agent how dangerous that enemy is. Kill probabilities are often useful for this decision. The kill probability of a given aircraft is the probability that one of its weapons could cause a kill.

Also included in this decision is the question of whether or not the opponent is interested. By calculating the overall threat of every aircraft, the one with the highest value is the most threatening. Thus an event should be generated within the agent notifying it of any high threat. This event may be masked to a given threshold, allowing the aircraft to select what threats need its attention. If the agent sets the threshold too high, it will not react to significant threats. If the threshold is set too low, however, the agent may be unable to accomplish its mission since every threat makes it react.

3.2.1.5 Aircraft Control. As I have previously stated, aircraft control is accomplished by the selection of a given action, or maneuver operator, in a combat situation. In other situations, one operator may be useful for many different things depending upon a variety of parameters. This helps to make each operator much more useful. These maneuvers then determine the amount of throttle and stick to use. By allowing the parameterization of maneuvers as well as their selection, fewer maneuvers need to be written. For example, the routine which causes the aircraft to cruise at a heading of 180 degrees and an altitude of 3,000 feet can also cause the aircraft to cruise at a heading of 90 degrees and 1,500 feet by changing two parameters. Thus all the agent needs to do to control the aircraft is to select an appropriate maneuver and set its parameters.

Not all maneuvers are of the type that continue forever. Some maneuvers will finish. Other maneuvers may fail. Although I have put together a sample maneuver architecture, it does not accomplish failure detection or recovery. This is left for later work. As a result, the maneuvers need to be very robust in order to keep from getting into dangerous situations. This poses some problems which will eventually need to be addressed as part of failure detection and recovery. More discussion of the actual implementation of these routines will follow in the next chapter.

3.2.2 Deliberative Reasoning. Both Shaw and Titan Systems identify several deliberative problems that need to be solved in flight (8)(13). In general, they fit into three broad categories. The first type is planning the aircraft's movements at a macro level. This type is concerned with avoiding certain ground-based hazards while meeting other aircraft goals. The second type is a form of planning on the micro level, as in what aircraft movements and maneuvers are appropriate and how should they be strung together. This form will attempt to solve the problem of short-range action look-ahead. The third category includes the quick, simple, decisions that pilots make during flight. These are the sorts of decisions which are not made continuously yet need extra thought. An example would be figuring out what the bingo fuel level should be for the mission, and then later recognizing when the aircraft was close enough to this fuel level to warrant returning to base. While the decision itself may not take all that much knowledge, being able to solve all such decisions represents a large quantity of essential knowledge. While this categorization is a sweeping generalization and there are bound to be deliberative problems that do not fit into any of these classifications, they are useful because they demonstrate the different types of autonomous solution techniques required. Of these three categories, only the first will be addressed in this effort.

Planning a path through an environment involves state-space search. Typically, the world is divided up into small cells and the agent is allowed to string them together searching for a path of least cost. As a metric, I have chosen to combine distance with visibility so that the agent searches for the shortest path with the least amount of visibility. The agent then uses the A* algorithm to find the cheapest path to its goal.

A couple of modifications are necessary to A*, however. First, since the agent needs contingency plans, the search starts from the goal and solves for the best path from every cell to the goal. (The dynamic programming principle makes this feasible.) By referencing the cell the agent is currently in, the agent has the answer of which cell to go to next. Second, in order to make this planning process adjust its solution as necessary, it repeats as often as possible. When, upon re-computation of a cell's cost it appears that a cheaper path is possible, then the plan is changed.

By letting the agent expand nodes from a list, the agent has control over how much time is spent processing the list. Thus, every time through the update cycle, the agent computes how many cells it has time to re-process. Then it expands that number of cells in order from the cheapest projected cost to the most expensive.

3.3 Experiment Design

Since the agent is tasked with both dog-fighting, as well as avoiding ground threats, several types of tests will be necessary to show its capabilities. The agent will be expected to demonstrate effective performance in both static environments as well dynamic ones. Two different types of basic problems will be addressed, which will exercise different portions of the simulation. The first is reactive combat, and the second is threat-avoidance.

The researchers from TacAir Soar tested their combat agents against three types of adversaries. These were non-jinking, jinking, and aggressive (9). A similar approach will be used to test the agents built in this effort. The first scenario will place an agent against a non-jinking drone. The second will place the agent against a drone that will jink. The third, and final, scenario will duel two identical agents against each other. In this manner, the agent first shows its capability in the static situations before attempting to demonstrate its abilities in the dynamic ones for comparison.

A similar approach will illustrate the capabilities of the integrated planner. First, the planner will be tested on a simple avoidance task, given that the entities it is attempting to avoid are stationary. After demonstrating its capabilities on such static threats, it will then tackle dynamic threats. This can be accomplished by adding drones to the simulation that fly loops over and around the battlefield. Together, these two forms of tests will demonstrate how well the aircraft can dynamically navigate the changing battlefield.

3.4 Summary

Presented in this chapter were several methods and techniques that have been used to reason in a dynamic combat environment. This reasoning design was based upon the necessity of timely responses in combat situations, and the necessity to spend more time reasoning about some problems (e.g. threat avoidance) than others. Finally, a method for demonstrating the capability of these routines was presented. The next chapter will discuss the implementation of the routines described here.

IV. Implementation

The design presented in the previous chapter has been implemented in C. Each part of this design will be described in detail in this chapter. The chapter starts out with a description of how the environment is maintained. Then, the method of aircraft control will be discussed, followed by descriptions of how the reaction and planning were accomplished. Each of these pieces adds an integral element to the simulation environment.

4.1 Environment Handling

Environment creation and maintenance starts at the top level. At this level, the simulation repeatedly reads the network and updates local models. These two tasks will be addressed in turn.

4.1.1 Network Reading. The manner of updating the external world is carried out in the network processing portion of the simulation. This portion, described in 3.1.1, has three components: a network reader, a dispatcher, and a database of all the active entities in the world.

The network reader reads every packet from a buffer the daemon keeps. Each packet is checked to make sure that it is part of the current exercise, and that its version is the same as the current DIS version (3). Once these two checks have passed, the packet is sent to a dispatcher. Upon completion of processing every packet in the daemon's buffer, this process returns control to the main program.

The dispatcher looks at the type of packet received. Then, based on the type, it filters the packet for relevancy and sends it to an appropriate processing routine. The most significant of these is the routine which maintains the world model, but also significant are the fire and detonation mechanisms. Detonations are sent to the agents they affect, while fire messages are routed to the agent's event mechanism (Section 4.2.2.4).

The world model is an array of the most recent ESPs from every entity. Upon time-out, or deactivation, entities are removed from the list. The agent can reference this list via requests for either an element of the list, or a member with a specific ID. Element members are returned in order from the beginning to the end of the list.

These processes, also described in 3.1.1, then maintain the information necessary for the agent to reason about the world around it.

4.1.2 Model Manager. The model manager may be viewed as part of the high-level system design. It maintains all of the local models that are currently active. There are three types of models: anti-aircraft-artillery pieces (AAAs), bullet bursts, and aircraft agents. Associated with each model are routines to update, pause, and damage it. There are also routines to create and destroy local models. Each model will be discussed shortly, in turn.

AAA is modelled simply: once they have been created and initialized with a position they remain in that same position. There, the AAA searches through the list of external entities for an enemy target which is visible and has not yet been destroyed. Upon finding the closest such target, the AAA will aim to kill. If it is currently aimed such that the bullet burst will come within ten meters of the enemy, it fires.

Bullet bursts are created by either an aircraft or an AAA firing upon an opponent. The firing entity will first attempt to target the burst and check how well it is currently aimed. Then, if the weapon is aimed and the player decides to fire, the burst is created flying towards its target. A fire PDU is issued so that all simulation participants may know the entity is firing. The burst will continue flying through the air as an entity, acted on only by gravity, until it either comes within a specified distance of the target or hits the ground. Intersection with the target is determined based upon how close the burst gets. Upon hitting a target or the ground, the burst broadcasts a detonation PDU and ceases to exist.

The final type of model in the simulation is the aircraft agent model. This models is the main component to the simulations. Upon update, the agent has a chance to reason and then updates its flight model. The details of this reasoning will be discussed later in this chapter. If the agent is damaged by a burst of bullets, however, the aircraft model determines how close the burst hit to dead center. If the burst hit the agent dead center, the agent is destroyed. If on the other hand the agent is hit further out, the damage decays exponentially. The aircraft, once damaged, responds less and less to its flight control inputs until finally, at one hundred percent damage, it turns into a falling piece of debris. Better damage models can be incorporated later as necessary.

Once a model is created, the simulation will update it as often as possible until it is destroyed. Upon update, the model manager calculates the time since the last update. It then updates each active model, giving it the time since the last update. Upon completion, all models have been updated to the current world state. The external world is then updated again, and the process repeats.

4.2 Phase processing

As discussed in Chapter III, the high-level aircraft state is represented by a phase. Although PDPC identified many phases to flight, I have found only six to be necessary to the simulation. These are take-off, cruise, attack, evade, avoid, and land.

4.2.1 Phase Descriptions. The *take-off* phase is perhaps the most simple of all phases. It begins by placing the aircraft into a peace-time flight envelope, and setting the current maneuver to *take-off.* This continues as the aircraft accelerates down the runway, until receipt of a *maneuver complete* event signifying that the aircraft has reached the necessary speed to take off. The agent then selects the *climb* maneuver and sets the altitude to be a thousand feet off the ground. When the aircraft starts to level off, and the agent receives another *maneuver complete* event, the agent transitions to the *cruise* phase.

Cruise phase is that phase used to get from place to place. Depending on the mission of the agent, the agent may follow a point-to-point script, or a predetermined path through the terrain. The agent leaves the cruise phase upon completion of the script, or upon a more deadly event such as a *high threat* event, a *new target* event, or a *missile warning.*

Attack phase is about attacking enemy aircraft. Upon entrance to this phase, the agent informs the weapons system to fire when aimed, and turns on the reactive maneuver selection process. The agent will leave this phase upon receiving a *target destroyed* event, a *missile warning* event, or a *change advantage* event.

If the opponent gains the advantage, the agent goes into *evade* phase. *Evade* is about reducing the opponent's firing advantage as quickly as possible. The reactive maneuver selection routines are used with a different state description table, and evasion maneuvers are selected instead of attacking maneuvers.

The *avoid* phase is similar to *attack* and *evade*, except that it is entered when another entity fires at the agent. The agent then tries to avoid the flying munition using a third state description table.

Finally, the *landing* phase is very similar to the *cruise* phase. The agent selects a landing pattern, and then executes the script. Upon completion of the script, the agent selects the *glide-slope* maneuver, followed by the *flare* maneuver, and finally the *ground-stop* maneuver. The maneuvers are changed whenever the agent receives a *maneuver complete* event. Then, upon landing, the agent is done with its mission and ceases to exist.

4.2.2 Event Types.

There are several types of events mentioned in Section 4.2.1. Specifically, there are seven types of events that the agent uses.

4.2.2.1 The Maneuver Complete Event.

This event is instantiated whenever the aircraft completes its current maneuver. This is determined by a flag set by the maneuver routine itself, so its meaning is highly dependent upon the current maneuver. For example, the *take off* maneuver is completed when the aircraft has reached take off speed. The *glide slope* maneuver, on the other hand, is complete when the aircraft is close enough to the ground to begin a flare.

4.2.2.2 The New Target Event.

For attacking aircraft, this is the event that is instantiated whenever the aircraft spots a target to chase after. This event causes a transition from *cruise* phase into *attack* phase, where the agent begins to pursue the enemy aircraft.

4.2.2.3 The High Threat Event.

Whenever the agent notices that another aircraft is tight on its tail, a *high threat* event is generated. This means that another aircraft is threatening the agent and the agent should react to it. Normally, this causes a transition to the *evade* phase.

4.2.2.4 The Munition Deployed Event. This event warns the agent that another entity has fired upon it. The agent will have a very limited amount of time to respond. This will cause the agent to transition to *avoid* phase.

4.2.2.5 The Change Advantage Event. This is used to determine whether the aircraft should transition from *attack* to *evade* or back again once an air-to-air target has been selected. Since the current agent is not very capable at picking its fights, the agent does not leave the air-to-air combat scene until its opponent is destroyed.

4.2.2.6 The Target Destroyed Event. This informs the agent that the target, munition, or enemy which it has been avoiding, evading, or attacking has been destroyed. It is a signal to return to the cruise phase.

4.2.2.7 The Null Event. Finally, the *null* event is the one generated when no other event needs processing.

Associated with each event is a priority level as well as a certain amount of data specific to the event. The priority levels are asserted upon creation of the event by the creator and are useful for determining which events to ignore (4.2.4).

4.2.3 Event Buffers. Since these events require different responses depending on when they are generated, and since ignoring an event may or may not cause it to go away, the storage of events is important. Four forms of event storage are provided: latest buffer, highest priority buffer, null queue, and a plain queue. Although none of the above events use a queue, I anticipate that further agent research will need to rely on this storage mechanism. It will be useful, perhaps necessary, to properly process agent-to-agent communication. The null queue produces a *null* event upon query, while the latest buffer and priority buffers just maintain the latest event or highest priority event of the given type. Therefore, each event type is associated with a buffer type that directs how the event and subsequent ones will be stored should they continue to occur.

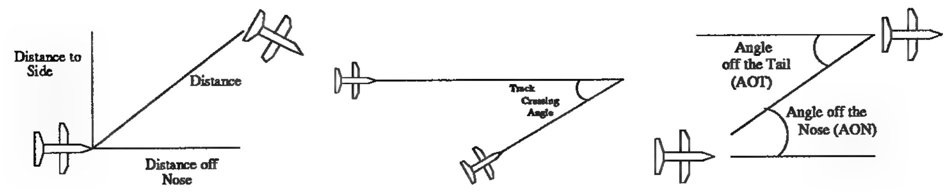

Figure 4.1 Six State Description Variables

4.2.4 Event Control. The agent has two basic methods to control how and if it receives events. The first is by masking events to specific priority levels. When the agent does this, events of a lower priority are ignored and neither placed in the buffer nor kept there. The other method is to reorder the events. Since the different buffers are checked in order until an event is found, re-ordering can have a profound effect. For example, by placing the null queue first, no other events will be received. Together, these two methods of event control help keep the aircraft focused on its current task or changing tasks.

4.3 Reaction

4.3.1 Threat Monitoring. Threat monitoring is a matter of looking at every other aircraft in the environment and determining how threatening or how vulnerable it is. This task runs along with the other ones, and will produce high threat and new target events together differing only by priority. The possibility is left open for future agents to use more complicated methods of threat-detection and selection in future work.

4.3.2 Maneuver Selection. Maneuver selection takes place in a couple parts. First, the agent selects the maneuver table to use. Then the agent traverses the table looking for the most appropriate maneuver.

Maneuvers are described by relationships to a predefined set of state descriptions. The state is described by ten variables. Figure 4.1 shows six of these variables. Other variables included in the state description are the energy advantage, and the rates of change for the three distance variables.

Each entry in the maneuver table consists of a evaluator, a variance, and a desired value. The five evaluators are less than, greater than, equal, not equal, and don't care (see

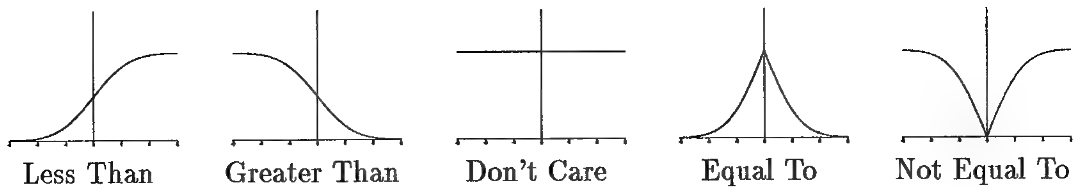

Figure 4.2 State Description Evaluators

Figure 4.2). These evaluators will compare two given numbers together, the measured value and the description value, and determine a fuzzy weighting for how close the description matches with the given amount of variance. The description evaluation results are then multiplied together. This result is applied to an overall maneuver utility to yield a score. The maneuver with the highest score is selected.

This manner of combining evaluation weights was borrowed from the field of probability. The evaluation functions *less than* and *greater than* are simply the integrated normal distribution and its inverse. Equals, and its inverse not equals, were created by applying the normal distribution to the absolute value of the difference, and multiplying by two to ensure the result added to one. However, although this method came from the field of probability, it was applied in an ad-hoc manner to this problem with the assumption that future work would clean up this process.

As part of maneuver selection, the agent looks for significant events in the simulation relating to maneuvers. For example, when the advantage changes a corresponding event is generated, as well as when the given adversary's existence is terminated. As a result, the maneuver selection routines deal with the majority of the air-to-air combat reactive processing once engagement has begun.

4.3.3 Weapon Utilization. Since only the aircraft cannon has been implemented, weapon utilization is quite simple. The agent informs the weapons system whether or not it may fire at the given opponent. Then, when the opponent is within range, the agent's weapon system fires until the opponent is destroyed. Future research will need to address the selection of multiple weapons, as well as the connection between the given weapon and the tactics needed to employ it.

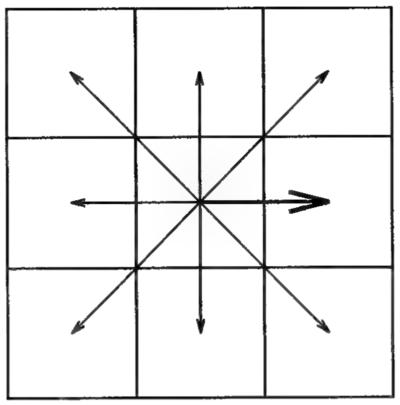

Figure 4.3 Possible Paths from a Cell

4.4 Agent Planning

As mentioned in Chapter III, the agent attempts to plan a path through a threat-filled environment as a deliberative process. The algorithm is A*, with a couple of modifications to allow for repeated use in a changing environment. The world is first divided into cells, in an array twenty by forty. Each cell is initialized to zero cost and a nowhere direction. Then the planner is applied to this cell-block repeatedly.

On update, the agent checks to make sure the simulation is not behind. If it is, the planner's time is forfeited to catch up. The planner then estimates the number of cells it can process based upon the amount of time which has elapsed. It creates this estimate by assuming that it can process 200 cells a second.

Cell information is maintained in a data structures containing a local cost, a cost to the goal, and a pointer to the next cell along the cheapest path to the goal. A cell may also have a flag identifying it as an illegal cell, notifying the agent to plan the quickest way out of any forbidden zones. This was how the agent avoided the mountain in the terrain since, due to the large distance between low-level sampling points, failure to do so often lead the agent through one of many mountain ridges. The cost of a cell is then determined by adding the number of enemy agents that may see the cell at any future time (N), times a constant (A), to the number that may almost see the cell (M) times another constant (B). The result is then incremented by one to insure that all cells have a minimum cost which is greater than zero.

$$Cost_{cell} = A * N + B * M + 1$$

The combining method, shown below, averages the individual costs of the two adjacent cells and multiplies the result by the distance (D) between them. This is the cost to travel from the current cell to the next cell. This partial cost is then added to the total cost from the next cell to the goal. The result is the total cost from the current cell to the goal.

$$Cost_{total} = \frac{Cost+Cost_{nextcell}}{2} * D + Cost_{total_{next\ cell}}$$

These methods were chosen for two reasons. First, they insure that the shortest path will always be the cheapest when it is completely invisible to all enemies. The constants, A and B, are the relationship between distance and threat. When $A = B = 0$, the agent will pick the shortest path. As A and B increase, the agent will pick longer and longer paths to avoid the given threats. Second, since the minimum cost between any two cells using this formula is the distance between them, distance can be used as an underestimator when determining which cells to update.

To process a cell, the cell is first selected from the list of cells to process. The cell with a minimum sum of total cost to goal and distance to the aircraft is selected. This is consistent with the definition of A*. The underestimator is the distance from the cell to the aircraft; the cost is the cell's current cost to the goal. The update recalculates the cell's local cost, and looks for the neighbor with the cheapest cost to the goal. That neighbor is selected as the next cell in the cheapest path, and the total cost is recalculated.

Once the new cost to the goal has been determined, the cell goes through each of its neighbors looking for any which need to be reopened. A cell needs to be opened if it has not yet been opened in the current update cycle, if there may be a cheaper path from that cell to the goal, or if its cheapest path runs through the cell just processed. By keeping track of which cells are on the list and which are not, the agent is also able to avoid placing a cell on the list twice.

The capabilities of the resulting planner will be discussed in detail in the next chapter.

4.5 Flight Control

Since this simulation depends on an actual aircraft flight model, an effort needs to be made to control this model. The model requires throttle and stick inputs in order to update its state. The agent, however, is not reasoning on this level. These routines, or maneuvers, discussed in the previous chapter will be defined here.

A maneuver is a function from the current state of the world to throttle and stick values. Since pilots 'know' how to fly, the goal of a maneuver is to capture the inner loop control that a pilot uses to accomplish a given set of goals. Maneuvers are memoryless. Since these maneuvers may be selected by the agent in an unpredictable manner, they also need to be robust. Most of the maneuvers are stateless, allowing for a simpler transition from maneuver to maneuver as necessary.

Since many of the maneuver functions require parameters, and since many of the parameters overlap, a shared data area is set aside for these. Parameters can be anything from altitude, speed, and heading, to leg, opponent, and runway. The parameters that are actually used will depend upon the maneuver selected.

4.5.1 Maneuvers.

The maneuvers can be separated into two different types; those that do not depend upon the position of another entity and those that do. Each is defined to accomplish a specific purpose, or goal, for the agent. Many will generate a *maneuver complete* event once that goal is accomplished.

4.5.1.1 The Take-Off Maneuver.

Like the *take-off* phase, this is the first and simplest maneuver. The aircraft basically holds the stick to neutral and the throttle to maximum until a given speed has been achieved. At this point, a *maneuver complete* event is generated so that the agent will recognize that it is time to continue with the mission.

4.5.1.2 The Climb Maneuver.

This maneuver holds a given heading while climbing to a preset altitude at a constant speed. After taking off, the agent generally transitions straight to the climb maneuver. This maneuver causes the aircraft to pitch up from its initial state and level off at its desired altitude. Even before level-off, however, a climb is complete when the aircraft is close enough to the given altitude to level out. At this time, the *maneuver complete* event is generated for the agent.

4.5.1.3 The Straight-and-Level Maneuver. The maneuver attempts to hold the aircraft in a steady flight state going straight and level. It is useful during transitions until the actual maneuver desired is known. This is the default action when the second entity in a two entity maneuver ceases to exist. A *maneuver complete* event is generated by this maneuver when the aircraft has achieved a steady flight state.

4.5.1.4 The Turn Maneuver. This maneuver turns the aircraft to the given heading while maintaining a preset altitude and speed. It generates a *maneuver complete* event upon reaching the desired heading.

4.5.1.5 The Cruise-To Maneuver. When the aircraft needs to fly to a given point, this maneuver will get it there. Normally, it flies straight to the point; however, this depends on altitude. If the altitude difference is great, it will fly up or down through the point. Otherwise, it will attempt to reach the altitude first and then the point. Once the point is behind the agent, a *maneuver complete* event is generated.

4.5.1.6 The Fly Leg Maneuver. This maneuver flies a point-to-point path, called a leg. Although the aircraft is trying to fly to the given end point, it will try to fly along the line between the beginning point and the ending point. This allows for the agent to calculate turn radii, round its turns, and then get back onto a flight path that it may not have been on originally.

4.5.1.7 The Glide Slope Maneuver. In order to approach a runway, the agent flies along a path leading it to the approach end of the runway. This leg has a given slope, and is aligned to the runway. Therefore this maneuver is a specialization of the *fly leg* maneuver.

4.5.1.8 The Flare Maneuver. This maneuver flies a leg along the runway, with a different altitude control. The objective is to touch the ground lightly, so a critically damped second order system is used to control altitude. This will take the aircraft directly from the glide slope maneuver to where it lightly touches the ground. A *maneuver complete* event is generated when the aircraft touches ground.

4.5.1.9 The Pursuit Maneuvers. *Lead*, *lag*, and *pure pursuit* maneuvers are each accomplished through one pursuit function. It picks a point to fly to in front of, on top of, or behind the opponent based on the desired lead angle. This angle is currently set at 10 degrees for lead, -10 degrees for lag, and 0 degrees for pure pursuit. This value is defined by Shaw (13) to be the angle between the aircraft's velocity vector and a line of site vector to the opponent. Since these routines are also used to set up gun shots, the offset or error vector from the gun is also added to the opponent's position to help the aircraft aim properly during pure pursuit.

4.5.1.10 The High YoYo. This routine is used to trade speed for altitude and a tighter turning radius to avoid overshooting an opponent. The aircraft climbs at the rate necessary to have a zero closure rate before it overshoots (13). This ensures that it can continue attacking its opponent upon completion without having lost any energy.

4.5.1.11 The Extension Maneuver. Given that the opponent has just overshot, and that the agent desires to disengage, this maneuver attempts to trade altitude for speed to get out of the current situation as soon as possible. It also tries to fly in the opposite direction as the opponent, but this is not as important as increasing the distance between the adversaries.

4.5.1.12 The Nose-to-Nose Turn Maneuver. This is the maneuver that produces head-to-head passes. The agent attempts to fly a leg parallel to the opponent's position and velocity, but closing.

4.5.1.13 The Missile Run Maneuver. Since the agent can turn hardest in the direction vertical to the cockpit, this maneuver attempts to place the missile straight above the cockpit. Then it pulls on the stick. Since there are no missiles in this simulation,

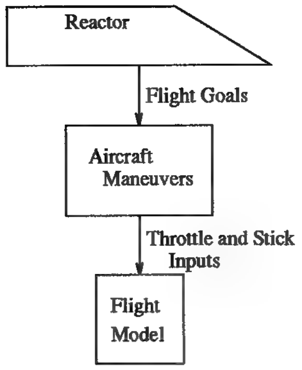

Figure 4.4 Maneuver Design

this routine is used to avoid bullet bursts instead. Its effectiveness is determined by how much time the agent has to change its motion while the bullet is in flight.

4.5.1.14 The Follow Map Maneuver. This maneuver is almost identical to the fly leg maneuver, except that instead of flying a given leg it flies from the cell it is in, with respect to the planner, to the next one on the current path..

4.5.2 Building Blocks. The reader may have noticed that many of the maneuvers share the same operators, with just different descriptions. Following a map and a glide-slope are two examples. Upon noticing that there were several things in common among the maneuvers, an attempt was made to extract the commonalities out to where they could be more useful. For example, flying a leg and flying to a point are both used in flying combat missions. These, however, are built upon another set of building blocks created to solve the more difficult problems. These are the altitude control, speed control, and turn control. The general flow of information through these routines can be seen in Figure 4.4.

4.5.2.1 Altitude Control. The altitude utility, which most of the maneuvers reference, creates a desired acceleration amount in the vertical direction. This amount depends on how far away the agent is from the desired altitude and how fast the agent is approaching the desired altitude. As such, it is a second order system.

When the agent is a long way from the desired altitude, it just climbs at some constant rate to reach that altitude. In this case, the acceleration desired is proportional to the difference between the desired and current climb-rates.

If the agent is closer, then a constant acceleration model is used. Instead of using a straight constant acceleration routine, however, the altitude utility calculates how far the aircraft is from the constant acceleration solution and adds or subtracts to the acceleration accordingly. This allows it to be more robust and to compensate for errors in this range.

Finally, when the agent is close to the desired altitude, a second order, critically damped, system is solved to determine the best acceleration. The constants are determined based on the maximum acceleration allowed by this system and the point where this solution and the constant acceleration one switch.

These three solutions will switch at points where they meet continuously. This keeps the aircraft from jumping around while it is trying to reach a given altitude. The two parameters controlling all three solutions are the desired climb rate and the maximum acceleration desired. From these two variables, the altitude utility can calculate which solution is appropriate for any flight condition. Together, these three solutions can accomplish very precise control of altitude during flight.

4.5.2.2 Speed Control. When the aircraft is not flying at its goal speed, it adds to the acceleration vector an amount proportional to the difference between desired and actual speeds and parallel to the aircraft's horizontal velocity. This does not work very well for near vertical flight; however, the rarity of this envelope keeps this routine useful.

4.5.2.3 Turn Control. The most complicated of all of the utilities is the turn utility. This routine adds to the acceleration vector an amount perpendicular to the current velocity and parallel to the ground. The amount is determined by first querying a database of aircraft capabilities to determine what is the tightest turn that the aircraft can perform without losing speed. Then, a roll angle for the turn is determined, as well as a roll-out angle. If the aircraft is within the roll-out angle of the desired direction, it attempts to roll out of the turn in a fashion such that the roll amount is proportional to how close the aircraft is to the desired heading. The roll rate is calculated to be proportional to the difference between the desired roll amount and the current roll amount.

There are two limitations to this technique. The first is that the agent cannot reason about complex tradeoffs between altitude, airspeed, and turn amounts. For example, turning really hard can cause the aircraft to slow down. However, if the aircraft is diving, it may be able to maintain that speed during the turn. The second limitation is that while this routine works for small angles, it does not attempt to fly the way a pilot would for small turn amounts. This is due to the stateless nature of the routine.

An illustration will help explain this difference. When a pilot needs to make a small turn of less than his maximum bank angle, the pilot will roll the number of degrees he is turning. Thus for a five degree turn, the pilot would roll five degrees. This depends upon knowing, when the turn is entered, that the turn will be only five degrees. The agent, on the other hand, is using the same routine to fly large angular turns as it is using to fly small turns. Thus when the agent has five degrees to turn, it does not know whether that is a five degree turn, or five degrees left in a sixty degree turn. It will attempt to solve both with the same solution. While this works, a side-effect is a large amount of roll used for small turns.

Despite the limitations, these utilities have worked successfully together to make flight control possible.

4.5.3 Throttle and Stick. Once the above utilities compute the acceleration and roll-rate for the agent to accomplish, a calculation routine converts these values to throttle and stick values. This routine works by calculating goal angles of attack and side-slip. The process assumes the power will be constant since throttle changes take time to take effect. A throttle setting is determined, based on the goal acceleration in the forward direction. Then the angle of attack and side slip angle are compared to the current values to produce yaw and pitch rates. These rates are applied to a Jacobian matrix to solve for appropriate stick amounts.

4.6 Summary

In the creation of an autonomous air combat simulation agent, many basic tasks need to be accomplished. This chapter has described these tasks and illustrated how they fit together and relate to each other. The next chapter, Analysis, will discuss how well this implementation solved the problems of air-to-air combat and threat avoidance.

V. Analysis

The agents were tested against the scenarios in Section 3.3. The results of these tests will be evaluated with respect to the agents' decision-making effectiveness and timeliness.

5.1 Effectiveness

5.1.1 Air-to-Air Combat. The agent was tested against three scenarios: a non-jinking bogey, a jinking bogey, and a fully reasoning agent. These opposing aircraft were agents whose offensive or defensive capabilities may have been turned off. The non-jinking drone, for example, is an agent that will not react to any threats. The jinking bogey, on the other hand, is an agent that will react to threats without attempting to take an offensive position. These three scenarios provide insight into the capabilities and mechanisms described in Chapter III.

5.1.1.1 Non-Jinking Bogey. The non-jinking bogey, or drone, demonstrated the greatest success that the agent had. The agent was able to move behind the drone quickly into a guns-firing position (see Figure 5.1). Upon reaching this position, the drone was destroyed.

This suggests that the basic pursuit maneuvers, which were used in this scenario, are very effective when the bogey is not jinking. Once a guns offset was added to the lead and pure pursuit routines, the agent was able to hit the target reliably. Without the offset this was not possible. Therefore the successful pursuit maneuvers were dependent upon the choice of weapons. This suggests that the appropriate maneuvers will need to change as different weapons are employed.

5.1.1.2 Jinking Bogey. The jinking bogey tested the capabilities of the agent much more thoroughly. The agent had great difficulty achieving a firing shot, depending on how often the drone jinked. When the drone held still, the agent quickly approached a firing solution as it had against the non-jinking bogey. If, during this process, the bogey jinked then the agent had to set the shot up again. When the bogey

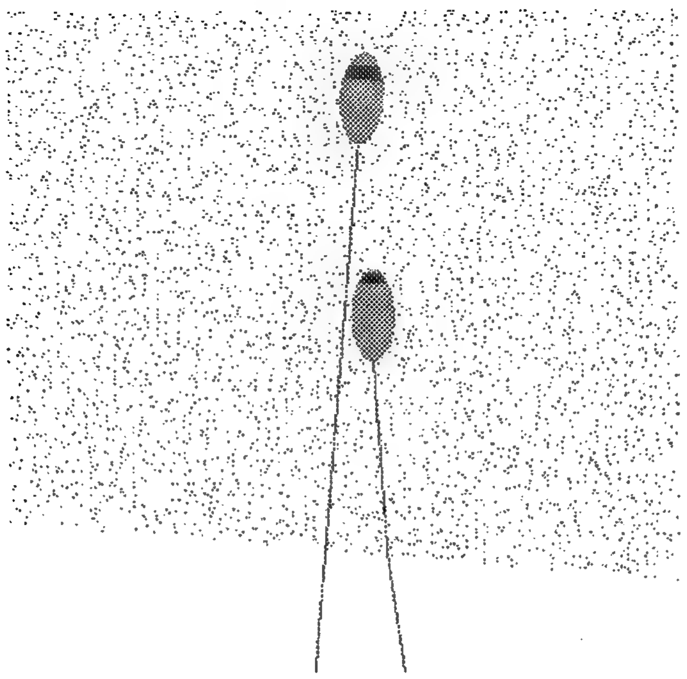

Figure 5.1 The Non-Jinking Bogey Scenario

maintainted a constant path, it was destroyed quickly as above. As soon as the drone moved significantly from its path, the agent overshot (see Figure 5.2).

According to Shaw, there are several ways to prevent such an overshoot (13). Two of these ways, the high yoyo and the lag-roll, were implemented. During the scenario, however, neither of these two maneuvers were used by the agent. Since the agent did not avoid the overshoot, it lost significant ground every time the drone made a hard turn.

This problem was caused by the lack of a complete knowledge representation of the state space. The representation was ideal for deciding between pursuit maneuvers. That decision is based primarily on whether the given angle off the tail is between 0 and 30 degrees (lead pursuit), 30 and 60 degrees (pure pursuit), and 60 and 90 degrees (lag pursuit). However, it did not convey the information necessary to represent the situation apparent when the agent was about to overshoot the drone. Shaw describes the time for a lag roll, as "approaching the target at close range with high overtake and low AOT (angle off the tail), ..." (13) This did not translate into any of the variables measured, despite the fact that both distance and its derivative were included. The distance derivatives, as

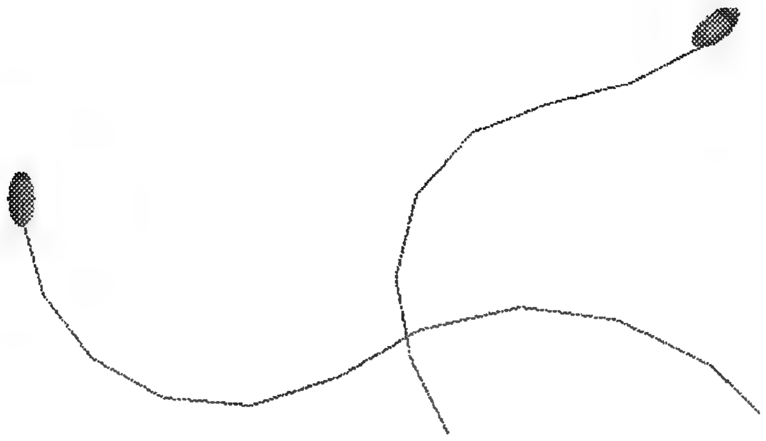

Figure 5.2 Overshoot against the Jinking Bogey

well as the distance variables, were inadequate in this case since the problem was a result of a combination of changing variables: the opponents changing direction, as well as the turn rates and radii of both agents. Such variable combinations could not be represented in the state description structure described in Section 4.3.2.

This structure was a success when it came to timeliness (see Section 5.3). The agent was able to quickly determine its situation, and meet the time-response criteria. This is something which PDPC, and the CLIPS-C code combination were unable to do. Because of this, it makes sense that the structure can be modified with the addition of new variables which will do a better job of describing the situations under which maneuvers are most effective.

The fact that the bogey does not move until the agent is in a very good firing position provides insight into the defensive capabilities of the bogey. Currently, some other form of determining which aircraft are threats is needed. An aircraft attempting to set up a shot should be considered a threat well before it gets into a firing position. The state description mechanism, which was used to determine this information, lacked the capability to combine variables in a complex manner. For example, how well another aircraft is aimed at a given aircraft is a complex function of the heading and pitch of the firing aircraft as well as the distance between aircraft. Since several decisions could have been made based upon this parameter, including it in the state description would be a reasonable idea.

Therefore, the agent passed this test with limited success. The pursuit routines, lead, lag, and pure, which are used during the overshoot are inappropriate for the situation leading to overshoot. Since the state descriptions did not convey the knowledge necessary to recognize the approaching overshoot, more information needs to be added.

5.1.1.3 Agent against Agent. When pitting two agents against each other, one of two results generally emerges. The first occurs when one agent begins with a significant advantage, and the second occurs when the agents begin with a head-to-head pass. When the agents begin from other angles, the result degenerates into one of these two situations.

When one agent begins with a significant advantage, if it is significant enough, the other agent is quickly destroyed before it has a chance to react. If it is not destroyed quickly, the scenario degenerates into the one where the agents originally started out with a head-to-head pass.

When the agents are started at a medium altitude in a head-to-head pass position, each aircraft begins a nose-to-tail turn after the pass. While no such maneuver exists explicitly, this is the result of applying the three pursuit routines to this situation. Then the agents turn around and around each other, maintaining constant positions on the opposite sides of the circle they are circumscribing. Their altitude drops continuously until they reach a given hard deck, while neither gains an advantage.

The agents reach an aerial combat stalemate. Shaw does not describe any similar scenarios (13). Two reasons will explain this apparent fault. The first is that the agents may not be intelligent enough, and the second is that the scenario may not be realistic enough.

It is easy to question the intelligence of the agent. Shaw describes several techniques for *baiting* the other pilot into a trap, but doing so means the aggressive agent must allow its opponent to gain a better angular advantage while the aggressive agent maintains greater energy. Such techniques were not programmed into the agents. As a result, neither

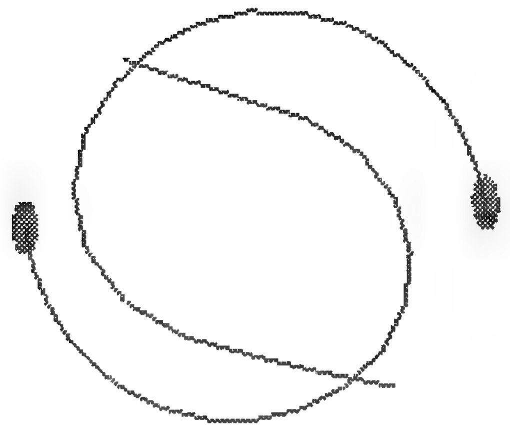

Figure 5.3 Agent vs Agent Stalemate

agent recognizes this possibility. A possible solution to this problem is to program the aircraft with other maneuvers and tactics that may be more profitable in these situations. By creating large sets of maneuvers/state-descriptions tactics can be created and reasoned about, allowing these situations to take place. Additional maneuvers, however, will require that the agent do a better job solving for the interplay between turn rates, speed, and altitude than the current method is capable of.

The other possibility is that the current simulation is not realistic enough to handle two identical agents combating against one another. Many adversary differences would keep this form of stalemate from happening. Possibilities include weight differences, airframe differences, and mission priority and goal differences. Therefore, it is suggested that future work consider adding these factors when combating aircraft agents against each other.

5.2 Threat Avoidance

The planner was tested with several scenarios. Two of them are shown in the appendix. The first forced the agent to plan around eight pop-up AAAs. The second forced the agent to plan around five AAAs and two drones. The evolution of the plan over time for each of these situations is shown in Figures A.1 through A.4 and Figures A.5 through A.9. All of the scenarios were engineered in such a way that if the aircraft continued on its original path it would be destroyed. The agent was forced to replan. Like the air-to-air combat, the solution effectiveness of this replanning degraded with the increasing dynamics of the situation. The faster things changed, the worse the agent performed.

In the scenario against the anti-aircraft-artillery, the aircraft did well when there were few AAA pieces. As the pieces increased, so did the time required from the beginning of the search to find the first solution. Once this solution was found, the aircraft was able to follow it without a problem. However, if the aircraft flew into the field before it had enough time to solve for a path, it was inevitably shot down after dodging a small number of bullets. This would be the scenario where the agent found out about a given threat only shortly before it came within range. This highlights the inadequacy of its purely reactive capability in this scenario.

When the drones were added, the agent performed well when the situation was simple as before. With a few AAAs and a few drones, the agent was capable of making it through. When the number of both increased, the result depended on the amount of prior planning time that the agent had. When the agent did not have prior time to plan its first path, it would get stuck in the far corner of the cell block attempting to find the first path. If it could find an initial path through the AAAs, it would successfully complete its path.

The planner had significant problems with something I will call a false positive. This is a plan built under a given set of preconditions which no longer held, being combined with a plan built with a different set of preconditions. Figure A.6 shows an example of this. The plan points the agent upwards in the cell block towards some cell. Initially, this cell had the cheap plan shown in Figure A.5. However, the agent continued building upon this plan once its preconditions were no longer valid. It also updated this original plan since its cheapest path no longer went straight back to the straight line distance in Figure A.5.

After the agent had initially planned paths throughout the whole block of cells, the total costs to the goal were kept in each of the cells. This was necessary to make sure that each local cell decision was able to pick the cheapest cost to the goal. A result of this is that it took time to propagate cost information. When a new threat appears, the cells updated near it compute a new total cost that is higher as a result of the threat. Distant cells maintain their previous decisions based upon the information that a cheap path led through the now threatened area (Figure A.6 again). The search mechanism then attempts to make choices about the best path closest to the aircraft, based upon inaccurate information. When the best path is found near the agent, the search expands outward towards the cells farther out that are left to be updated. Once the agent discovers that the current path is not the best, it plans for the cheapest short-term solution again. The resulting search process resembles a poor depth-first search algorithm. The partial solutions, while better than none, changes so rapidly that the agent could not be viewed as following a consistent plan since the best next cell kept changing as the agent tried to fly to it. (Note how the agent, between Figure A.2 and A.3 has completely changed direction.)

Despite these shortfalls, however, the agent did manage to replan a safe route through its world. This was not possible through the reactive mechanism which would have been appropriate in a world without threats. Under this circumstance, the agent would fly straight into the range of the first AAA site and be destroyed. Only by planning a path around that site and others was it capable of successfully navigating the space.

5.3 Solution Timing

This area demonstrates the biggest success of the reaction solutions, and the biggest failures of the planner.

The reactor was able to maintain the tenth of a second limit placed upon it when other processes were not loading down the CPU. However, the CPU load tended to fluctuate often. As a result, while the agent usually met its timing requirements it did not do so reliably. In order to keep this effect from accumulating, it was necessary to insure that there was a sufficient amount slack time in the agent's processes.

When the question turns to re-planning times, the answer changes. The agent was capable of processing about 200 cells per second. It also took some time to initially solve for a complete solution. The fact that distance is not the best under-estimator when visibility is the primary cost component can contribute to this problem. Once the first solution was found, however, it was normally kept up to date by only checking between 800 and 900 cells, depending on the changes in the environment. The extra cell processing, above the 800 cells in the space, is due to the necessity of updating and re-updating paths whose preconditions changed. For example, a cell will be re-evaluated if an adjacent cell calculates a cheaper cost than it originally had. Significant environment changes usually caused updates in the ten and twenty thousand cell ranges; however since the planner could degenerate into a depth first search the number of cells needing to be updated could reach as high as forty thousand cells. Since this may take more time than the agent has, the agent is forced to use the partial solution mixed with its prior solution. The quality of that solution may not be the quality necessary or desired.

5.4 Summary

This chapter has presented the results of several combat scenario tests. Its limited effectiveness in the air combat domain, and the timeliness of its answers, suggests that minor modifications to the state descriptions and additional maneuvers will make the agent much more capable. Its accomplishments in the threat avoidance domain demonstrate that it is possible to integrate planning and reaction together within one agent. The next chapter, Conclusions, will suggest future study efforts in this field.

VI. Conclusions

Several separate conclusions can be drawn from this work.

First, intelligent agents can successfully control complex objects by selecting from among control routines and setting their parameters. This method proved to be a very effective method of controlling the aircraft, allowing for precise control in predetermined situations. Further work in this area, however, is necessary to expand this area of knowledge to include more knowledge of maneuver failure and failure states. This will be necessary if agents are to accurately model all of the behaviors required in a combat scenario, including the handling of an aircraft that responds poorly. It is also necessary before this method will be reliable enough to fly multi-million dollar pieces of equipment.

Second, reaction can be a timely reasoning method in highly dynamic situations. If the agent does not have the time to plan, some decision still needs to be made. By using reactive mechanisms, the agents developed in this effort were capable of timely solutions to problems in their environment. Further work is necessary in the area of fighter combat reaction to create better state descriptions of the air-to-air combat battlefield from which to react (see Section 5.1.1.2).

Third, the planner demonstrated that long-term planning increased the chance of mission success. If the agent attempted to follow its initial plan through the threats, it was destroyed. By re-planning its path in flight, it was able to calculate a path through the threat environment without being destroyed. This would not have been possible through a simple reactive mechanism. Future work should focus on exactly how this long-term planning should be accomplished, as well as how it can be tuned to work consistently in a dynamic environment.

Finally, it is possible to integrate reaction and deliberation in a real-time scenario. Reaction by itself is not enough in a combat scenario (2). This simulation presented scenarios where the agent would be destroyed if it stayed with its purely reactive strategy (see Section 5.2). By integrating planning with reaction the agent was more capable than it would have been with reaction only.

This research has demonstrated, therefore, that agents can fly complex objects through reaction, and that the integration of a planner can result in better performance. Future work in the air-to-air combat domain should focus on filling in the current gaps. Among them, the need for a better state description language, a larger set of maneuvers, and a broader planning ability are apparent in the results of this effort.

Appendix A. Planning Partial Solutions

The following diagrams outline the evolution of an agent's plan as it attempts to fly from the left side of the page to the right. The arrows located within each cell are the agents current best action choices, and the heavy line follows the agents path from its location to the goal. In each scenario, the agent was allowed to develop the initial map before the threats were added.

Figures A.1 through A.4 outline the agent's attempts to dynamically plan around eight AAA sites.

Figures A.5 through A.9, on the other hand, show the agent's progress as it avoids two drone aircraft and five AAA sites. The last two figures, A.8 and A.9, show how the agent ended up when it needed to avoid the drones. An example of the false positive problem can be seen in Figure A.6.

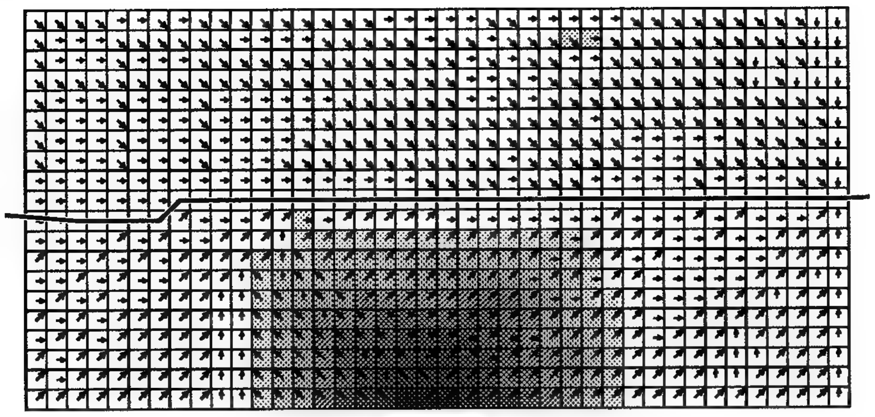

Figure A.1 Agent vs AAAs, T=0 min

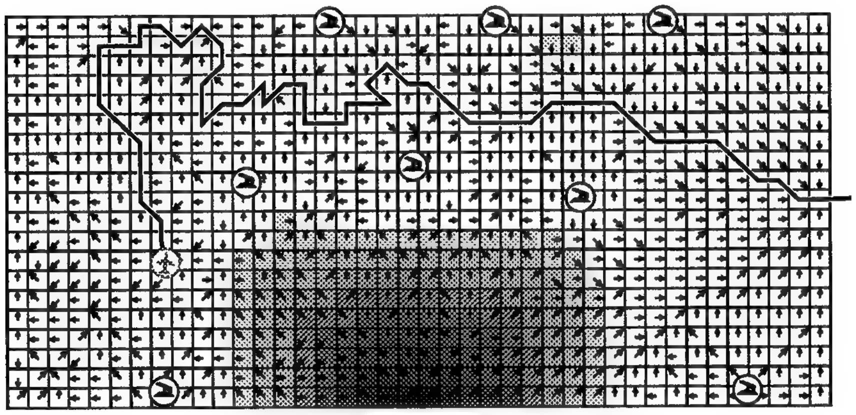

Figure A.2 Agent vs AAAs, T=2 min

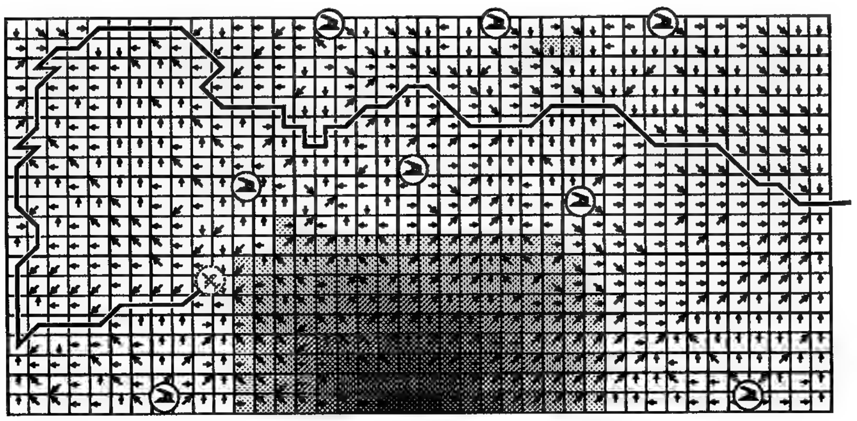

Figure A.3 Agent vs AAAs, T=4 min

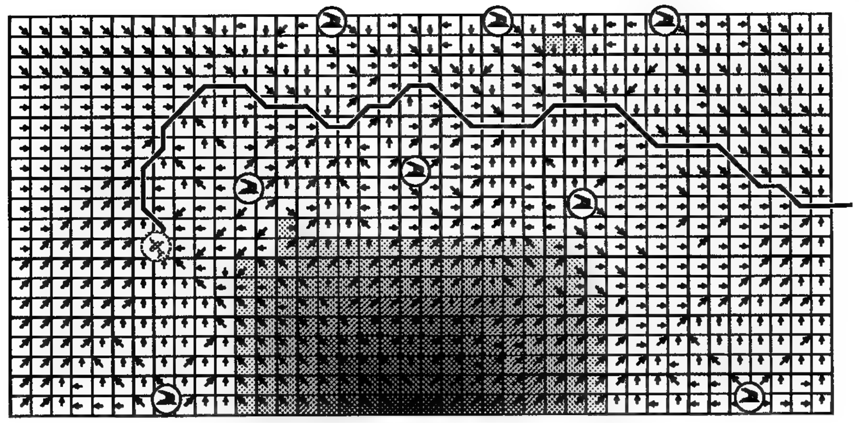

Figure A.4 Agent vs AAAs, T=6 min

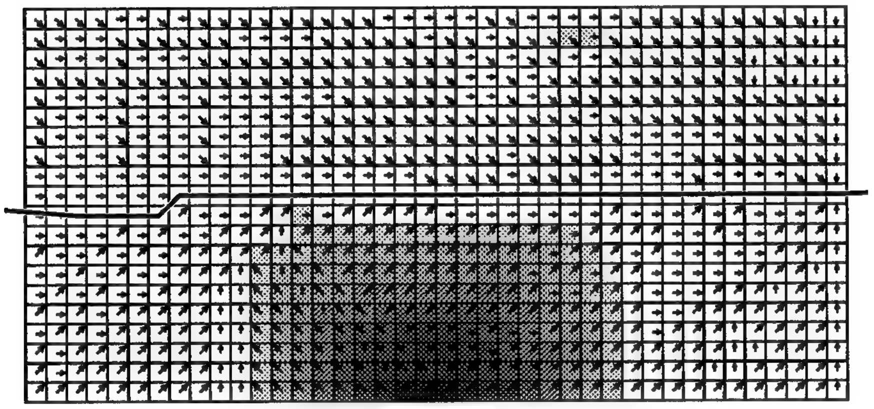

Figure A.5 Agent vs AAAs and Drones, T=0 min

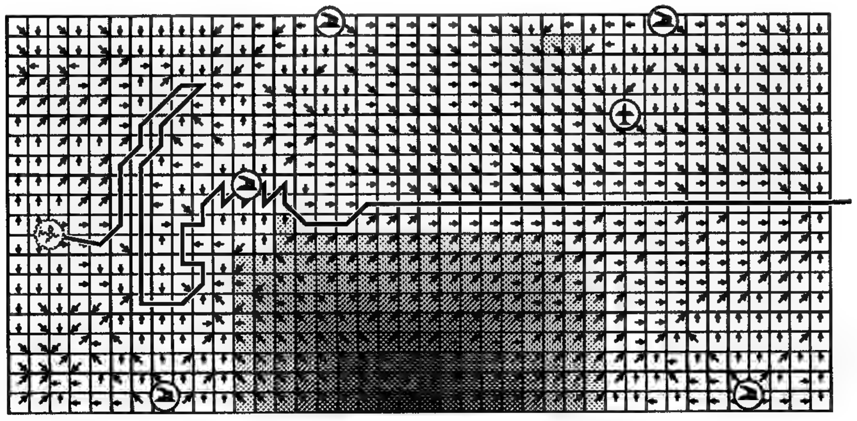

Figure A.6 Agent vs AAAs and Drones, T=1 min

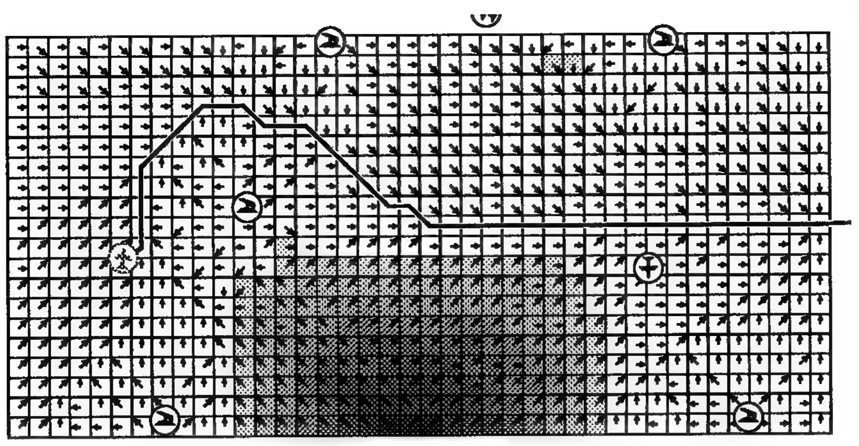
Figure A.7 Agent vs AAAs and Drones, T=2 min

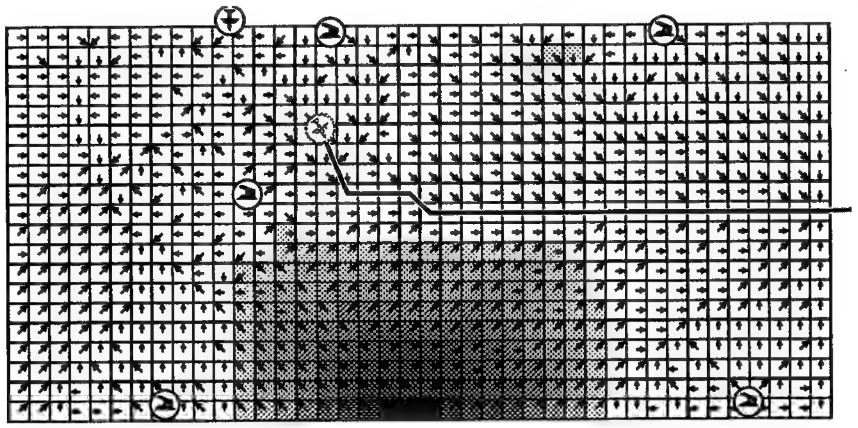
Figure A.8 Agent vs AAAs and Drones, T=4:30 min

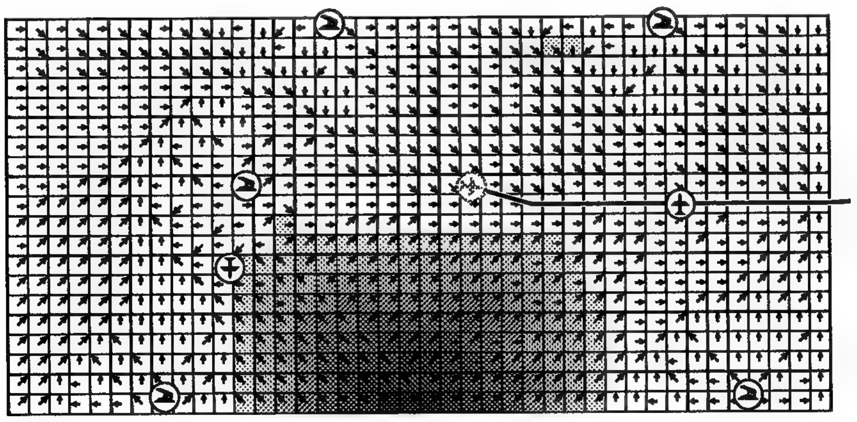
Figure A.9 Agent vs AAAs and Drones, T=5:40 min

Vita

Second Lieutenant Daniel Gisselquist was born in Austin, Texas on 5 July, 1971. He graduated from Bloomington Kennedy Sr. High in Minnesota after spending his senior year at St. Olaf college in Northfield, Minnesota. He was accepted into the USAF Academy in 1989, and graduated in 1993 with a major in mathematics and a major in computer science.

Permanent address: 10632 Upton Av S
Bloomington, MN 55431

Bibliography

1. Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence.*, July 1987.
2. Douglas E. Dyer and Gregg H. Gunsch. Enlarging the universal plan for air combat adversaries. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation.*, pages 255–261, March 1993.
3. Institute for Simulation and Training. Standard for information technology–protocols for distributed interactive simulation applications. Proposed IEEE Standard Draft IST-CR-93-15, University of Central Florida, 1993.
4. Melinda T. Gervasio. Learning general completable reactive plans. In *Proceedings of the American Association for Artificial Intelligence*, pages 1016–1021, 1990.
5. Maj Gregg H. Gunsch, Capt. Douglas E. Dyer, Capt. Mark J. Gerken, Capt. Laurence D. Merkle, and Capt. Michael A. Whelan. Autonomous adversaires for interactive air combat simulation. Sponsored by DARPA.
6. Dean P. Hipwell. Developing realistic cooperative behaviors for autonomous agents in air combat simulation. Master's thesis, Graduate School of Engineering of the Air Force Institute of Technology, 1993.
7. George S. Hluck. Developing realistic behaviors in adversarial agents for air combat simulation. Master's thesis, Graduate School of Engineering of the Air Force Institute of Technology, 1993.
8. Titan Systems Inc. Pilot's decision definition and analysis. Technical Report AFWAL-TR-86-0002, Air Force Wright Aeronautical Laboratories, 1986.
9. Randolph M. Jones, Milind Tambe, John E. Laird, and Paul Rosenbloom. Intelligent automated agents for flight training simulators. In *Third Conference on Computer Generated Forces and Behavior Representation*, 1993.
10. Tom M. Mitchell. Becoming increasingly reactive. In *Proceedings of the American Association for Aritificial Intelligence*, pages 1051–1058, 1990.
11. Paul S. Rosenbloom, John E. Laird, Allen Newell, and Robert McCarl. A preliminary analysis of the Soar architecture as a basis for general intelligence. *Artificial Intelligence*, 47:289–325, 1991.
12. M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence.*, 1987.
13. Robert L. Shaw. *Fighter Combat: Tactics and Maneuvering.* Naval Institute Press, 1985.
14. Shashi Shekhar and Soumitra Dutta. Minimizing response times in real time planning and search. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 1989.
15. Brian L. Stevens and Frank L. Lewis. *Aircraft Control and Simulation.* John Wiley & Sons, Inc., 1992.

16. Milind Tambe and Randolph M. Jones. Overview of TacAir-Soar. In *Thirteenth Soar Workshop*, pages 207–209, March 1994.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE
ARTIFICIALLY INTELLIGENT AIR COMBAT SIMULATION AGENTS

5. FUNDING NUMBERS

6. AUTHOR(S)
Daniel E. Gisselquist

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCE/ENG/94D-04

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Advanced Systems Research Section
Information Processing Technology Branch
System Avionics Division
Avionics Directorate
Wright Laboratories

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
This work is concerned with developing techniques that may be used in the creation of artificially intelligent air combat simulation agents. It demonstrates, by example, that it is possible to integrate both reactive and deliberative behaviors within a highly dynamic real-time combat environment. The culmination is an intelligent agent which exhibits many of the behaviors necessary in such a combat situation.

14. SUBJECT TERMS
Air Combat Simulation, Artificial Intelligence, Planning, Reaction

15. NUMBER OF PAGES
67

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL